Тема 2.1 От С к С++

По большей части С++ представляет собой надстройку над стандартным ANSI C, и фактически все программы на С являются также программами и на С++. Тем не менее между этими языками имеется несколько отличий, и наиболее важные из них обсуждаются ниже.

Отличие 1. Исключения

Исключение по своей сути - это просто последовательность goto и return. Основан на обычной С-технологии setjmp/longjmp. try и catch - это setjmp с проверкой. throw - это longjmp. Когда вызывается throw, то проверяется: если он окажется внутри блока try, то выполняется goto на парный блок catch. Если нет, то делается return и ищется catch на уровень выше и так далее.

Наличие в throw/catch параметра ничего принципиально не меняет: и в обычном С можно было заполнить какие-то переменные перед вызовом longjmp и потом их проанализировать.

Отличие 2. Перегруженные операторы

Пример функции a + b, где a и b - типа Point это функция от двух аргументов a и b, возвращающая Point:

```
Point operator+(Point a, Point b)
```

Написать a+b равносильно вызову такой функции: operator+(a,b). Иногда эта технология удобна, а иногда вносит путаницу.

Отличие 3. Ссылка

Многие программисты изучали С на основе языка Pascal. В Pascal есть возможность возвращать из функции больше одного параметра. Для этого применялось слово "var". В С для того, чтобы сделать то же самое, приходилось расставлять в тексте достаточно большое количество символов "*".

Разработчики С++ учли пожелания программистов и ввели символ "&" - аналог var и назвали его "ссылкой". Это вызвало большую путаницу, так как в С уже были понятия "указатель" (та самая звездочка) и "адрес" (обозначался тем же символом &), а понятие "ссылка" звучит тоже как что-то указующе-адресующее.

С одной стороны, использование ссылок намного сокращает текст программы. Но есть и неприятности. Во-первых, вызов функции, в которой параметр является ссылкой, выглядит так же, как вызов с обычным параметром. В результате "на глаз" незаметно, что параметр может измениться. А в C это заметно по значку &. Во-вторых, многочисленные звездочки в C напоминают программисту о том, что каждый раз выполняется дополнительная операция * разыменования указателя. Что побуждает сделать разумную оптимизацию. В C++ эти операции остаются незамеченными.

Отличие 4. Глобальная область видимости

Как бы ни была хороша локальность имён, разные подзадачи, в совокупности композируемые в одну большую задачу, должны как-то обмениваться информацией. И делать это средствами только посредством имён из локальных областей видимости самих подзадач невозможно. Поэтому вполне естественно, что они это делают посредством окаймляющих областей видимости. Если разные локальные области видимости имеют для себя одну и ту же, непосредственную или опосредованную, окаймляющую область видимости, именно имена из неё могут использоваться для взаимодействия подзадач друг с другом.

Понятно, что матрёшка областей видимости не может простираться бесконечно во вне. Подзадачи в конце концов всегда являются кирпичиками некой одной большой задачи, следовательно и дальнейшее окаймление областей видимости, начиная с некоего уровня, становится ненужным, ибо выше уже нет подзадач, требующих взаимодействия. Так что где-то обязательно должен находиться абсолютный предел, выше которого окаймляющих областей видимости уже нет. Именно таким пределом и является глобальная область видимости. И она тоже не составляет исключения из замеченного выше свойства: её характеристики почти одинаковы в обоих языках.

Методы декомпозиции в C более ограничены, нежели в C++. Структуры как таковые являются просто контейнером, группирующим имена переменных (называемые в этом контексте полями структуры) в нечто цельное, удобное для манипулирования ими всеми одновременно и инкапсулирующими логически связанные характеристики в единую сущность, рассматриваемую как цельную. Собственно это и всё.

Итоги:

Конечно же, проблемы, связанные с ошибками в компиляторах, проявляются очень редко. Собственно поэтому можно уже сейчас оценить круг задач, которые лучше решать при помощи C++, чем C (при наличии, конечно же, хороших навыков программирования в обоих языках): это практически все программы, от которых не требуется беспрерывная работа 24 часа в сутки. Очень неприятно обнаружить, что программа, которая писалась и отлаживалась на каких-то тестовых примерах, не может выдержать реальной нагрузки и проблема кроется именно в том, что где-то глубоко внутри библиотеки, поставляемой с компилятором, не был реализован механизм блокировки доступа к разделяемому ресурсу. Кроме того, обычно переносимость программы с одного компилятора на другой уменьшает количество используемых возможностей языка программирования, потому что разные компиляторы, как это ни смешно звучит, по разному "соответствуют стандарту". Или, точнее, не соответствуют ему. А подобное ограничение на конструкции языка (одно из самых обидных лишений, конечно же, ограничение на использование шаблонов) сводит на нет большинство преимуществ C++.

В таких случаях выбор языка программирования C вместо C++ более предпочтителен, так как даст возможность изначально уменьшить количество непонятных проблем, возникающих в реальной эксплуатации программного продукта.

При этом всём, стоит отметить потенциальную опасность C, который традиционно позволяет программисту делать все что угодно, зачастую пропуская его ошибки. Но эти ошибки выловить иногда значительно легче, чем объяснять различные странности, появляющиеся то тут, то там в программах на C++.

Более строго, можно сказать, что обычно к разрабатываемому программному обеспечению предъявляются какие-то требования, связанные с его качеством. Эти требования не могут быть настолько жесткими, чтобы совсем исключать вероятность наличия ошибок в программе, но чем они сильнее, тем лучше использовать старый и проверенный во многих разработках C. В остальных же проектах может быть обратная ситуация: сложность разработки на C вызовет увеличение сроков, связанное с отладкой и выявлением ошибок в коде у самих программистов.